100个gcc小技巧

wizardforcel

Published
with GitBook



目錄

《100个gcc小技巧》

> 作者：hellogcc
>
> 来源：100-gcc-tips

一个关于gcc使用小技巧的文档。100，在这里可能只是表明很多；具体的数目取决于您的参与和贡献。

在线阅读

开始阅读

如何参与

直接发PULL REQUEST，或与我们联系。

增加一个小技巧的步骤：

1. 在src目录下新增一个md文件，参照现有文件的格式风格，编写一个小技巧
   markdown语法参见 http://wowubuntu.com/markdown/
   md文件编写可以使用在线所见即所得编辑器
   https://www.zybuluo.com/mdeditor
2. 在index.md中为新md文件增加一个索引，可以放到已有分类中，或增加一个分类
3. 如果预览下没有问题，OK!

本地生成html的步骤：

1. 确保go和md2min已经安装并可用
2. 直接运行build.sh
3. 如果顺利，会在html目录下生成所有的html文件

联系方式

- 博客网站
- 在线讨论问题：IRC, freenode, #hellogcc房间
- 邮件列表 (发信需要先订阅)

版权

授权许可

致谢

- 各位参与者

其它资源

- GCC在线手册

信息显示

打印gcc预定义的宏信息

例子

```
[root@linux:~]$ gcc -dM -E - < /dev/null
#define __DBL_MIN_EXP__ (-1021)
#define __FLT_MIN__ 1.17549435e-38F
#define __CHAR_BIT__ 8
#define __WCHAR_MAX__ 2147483647
#define __GCC_HAVE_SYNC_COMPARE_AND_SWAP_1 1
#define __GCC_HAVE_SYNC_COMPARE_AND_SWAP_2 1
#define __GCC_HAVE_SYNC_COMPARE_AND_SWAP_4 1
#define __DBL_DENORM_MIN__ 4.9406564584124654e-324
#define __GCC_HAVE_SYNC_COMPARE_AND_SWAP_8 1
#define __FLT_EVAL_METHOD__ 0
#define __unix__ 1
#define __x86_64 1
#define __DBL_MIN_10_EXP__ (-307)
#define __FINITE_MATH_ONLY__ 0
#define __GNUC_PATCHLEVEL__ 7
```

技巧

如上所示，使用“`gcc -dM -E - < /dev/null`”命令就可以显示出gcc预定义的宏信息。“`-dM`”生成预定义的宏信息，“`-E`”表示预处理操作完成后就停止，不再进行下面的操作。此外，也可以使用这个命令：“`echo | gcc -dM -E -`”。

详情参见gcc手册

贡献者

nanxiao

打印gcc执行的子命令

例子

```
$ gcc -### foo.c
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/4.6/lto-wrapper
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu/Lina
Thread model: posix
gcc version 4.6.3 (Ubuntu/Linaro 4.6.3-1ubuntu5)
COLLECT_GCC_OPTIONS='-mtune=generic' '-march=x86-64'
 /usr/lib/gcc/x86_64-linux-gnu/4.6/cc1 -quiet -imultilib . -imultia
COLLECT_GCC_OPTIONS='-mtune=generic' '-march=x86-64'
 as --64 -o /tmp/cc9Ce7IE.o /tmp/ccezMraJ.s
COMPILER_PATH=/usr/lib/gcc/x86_64-linux-gnu/4.6/:/usr/lib/gcc/x86_6
LIBRARY_PATH=/home/xmj/install/cap-llvm-3.4/lib/../lib/:/usr/lib/gc
COLLECT_GCC_OPTIONS='-mtune=generic' '-march=x86-64'
 /usr/lib/gcc/x86_64-linux-gnu/4.6/collect2 "--sysroot=/" --build-i
```

技巧

如上所示，使用`-###`选项可以打印出gcc所执行的各个子命令，分别为，

cc1：

```
 /usr/lib/gcc/x86_64-linux-gnu/4.6/cc1 -quiet -imultilib . -imultia
```

as：

```
 as --64 -o /tmp/cc9Ce7IE.o /tmp/ccezMraJ.s
```

collect2：

```
 /usr/lib/gcc/x86_64-linux-gnu/4.6/collect2 "--sysroot=/" --build-i
```

这个跟使用`-v`所显示的内容差不多，区别在于使用`-###`是只打印，不实际执行具体的命令。手册里提到，它的一种用法，就是在脚本里使用这个选项，来获得gcc所调用的各个子命令行。

详情参见gcc手册

贡献者

xmj

打印优化级别的对应选项

例子

```
$ gcc -Q --help=optimizers
The following options control optimizations:
  -O<number>
  -Ofast
  -Os
  -falign-functions                   [disabled]
  -falign-jumps                       [disabled]
  -falign-labels                      [disabled]
  -falign-loops                       [disabled]
  -fasynchronous-unwind-tables         [enabled]
  -fbranch-count-reg                  [enabled]
  -fbranch-probabilities              [disabled]
  -fbranch-target-load-optimize     [disabled]
  -fbranch-target-load-optimize2     [disabled]
  -fbtr-bb-exclusive                  [disabled]
  -fcaller-saves                      [disabled]
  -fcombine-stack-adjustments         [disabled]
  -fcommon                            [enabled]
  -fcompare-elim                      [disabled]
  -fconserve-stack                    [disabled]
  -fcprop-registers                   [disabled]
  -fcrossjumping                      [disabled]
  -fcse-follow-jumps                  [disabled]
  -fcx-fortran-rules                  [disabled]
  -fcx-limited-range                  [disabled]
  -fdata-sections                     [disabled]
  -fdce                               [enabled]
  -fdefer-pop                         [disabled]
  -fdelayed-branch                    [disabled]
  -fdelete-null-pointer-checks         [enabled]
  -fdevirtualize                      [disabled]
  -fdse                               [enabled]
  -fearly-inlining                    [enabled]
  -fexceptions                        [disabled]
  -fexpensive-optimizations           [disabled]
  -ffinite-math-only                  [disabled]
  -ffloat-store                       [disabled]
  -fforward-propagate                 [disabled]
  -fgcse                              [disabled]
  -fgcse-after-reload                 [disabled]
  -fgcse-las                          [disabled]
  -fgcse-lm                           [enabled]
  -fgcse-sm                           [disabled]
  -fgraphite-identity                 [disabled]
```

```
  -fguess-branch-probability          [disabled]
  -fhandle-exceptions
  -fif-conversion                     [disabled]
  -fif-conversion2                    [disabled]
  -finline-functions                  [disabled]
  -finline-functions-called-once     [enabled]
  -finline-small-functions            [disabled]
  -fipa-cp                            [disabled]
  -fipa-cp-clone                      [disabled]
  -fipa-matrix-reorg                  [disabled]
  -fipa-profile                       [disabled]
  -fipa-pta                           [disabled]
  -fipa-pure-const                    [disabled]
  -fipa-reference                     [disabled]
  -fipa-sra                           [disabled]
  -fivopts                            [enabled]
  -fjump-tables                       [enabled]
  -floop-block                        [disabled]
  -floop-flatten                      [disabled]
  -floop-interchange                  [disabled]
  -floop-parallelize-all              [disabled]
  -floop-strip-mine                   [disabled]
  -flto-report                        [disabled]
  -fltrans                            [disabled]
  -fmath-errno                        [enabled]
  -fmerge-all-constants               [disabled]
  -fmerge-constants                   [disabled]
  -fmodulo-sched                      [disabled]
  -fmove-loop-invariants              [enabled]
  -fnon-call-exceptions               [disabled]
  -fnothrow-opt                       [disabled]
  -fomit-frame-pointer                [disabled]
  -foptimize-register-move            [disabled]
  -foptimize-sibling-calls            [disabled]
  -fpack-struct                       [disabled]
  -fpack-struct=<number>
  -fpeel-loops                        [disabled]
  -fpeephole                          [enabled]
  -fpeephole2                         [disabled]
  -fpredictive-commoning              [disabled]
  -fprefetch-loop-arrays              [enabled]
  -freg-struct-return                 [disabled]
  -fregmove                           [disabled]
  -frename-registers                  [enabled]
  -freorder-blocks                    [disabled]
  -freorder-blocks-and-partition     [disabled]
  -freorder-functions                 [disabled]
  -frerun-cse-after-loop              [disabled]
  -freschedule-modulo-scheduled-loops     [disabled]
  -frounding-math                     [disabled]
  -frtti                              [enabled]
  -fsched-critical-path-heuristic     [enabled]
  -fsched-dep-count-heuristic         [enabled]
```

```
  -fsched-group-heuristic                [enabled]
  -fsched-interblock                     [enabled]
  -fsched-last-insn-heuristic            [enabled]
  -fsched-pressure                       [disabled]
  -fsched-rank-heuristic                 [enabled]
  -fsched-spec                           [enabled]
  -fsched-spec-insn-heuristic            [enabled]
  -fsched-spec-load                      [disabled]
  -fsched-spec-load-dangerous            [disabled]
  -fsched-stalled-insns                  [disabled]
  -fsched-stalled-insns-dep              [enabled]
  -fsched2-use-superblocks               [disabled]
  -fschedule-insns                       [disabled]
  -fschedule-insns2                      [disabled]
  -fsection-anchors                      [disabled]
  -fsel-sched-pipelining                 [disabled]
  -fsel-sched-pipelining-outer-loops        [disabled]
  -fsel-sched-reschedule-pipelined        [disabled]
  -fselective-scheduling                 [disabled]
  -fselective-scheduling2                [disabled]
  -fshort-double                         [disabled]
  -fshort-enums                          [enabled]
  -fshort-wchar                          [disabled]
  -fsignaling-nans                       [disabled]
  -fsigned-zeros                         [enabled]
  -fsingle-precision-constant            [disabled]
  -fsplit-ivs-in-unroller                [enabled]
  -fsplit-wide-types                     [disabled]
  -fstrict-aliasing                      [disabled]
  -fstrict-enums                         [disabled]
  -fthread-jumps                         [disabled]
  -fno-threadsafe-statics                [enabled]
  -ftoplevel-reorder                     [enabled]
  -ftrapping-math                        [enabled]
  -ftrapv                                [disabled]
  -ftree-bit-ccp                         [disabled]
  -ftree-builtin-call-dce                [disabled]
  -ftree-ccp                             [disabled]
  -ftree-ch                              [disabled]
  -ftree-copy-prop                       [disabled]
  -ftree-copyrename                      [disabled]
  -ftree-cselim                          [enabled]
  -ftree-dce                             [disabled]
  -ftree-dominator-opts                  [disabled]
  -ftree-dse                             [disabled]
  -ftree-forwprop                        [enabled]
  -ftree-fre                             [disabled]
  -ftree-loop-distribute-patterns        [disabled]
  -ftree-loop-distribution               [disabled]
  -ftree-loop-if-convert                 [enabled]
  -ftree-loop-if-convert-stores        [disabled]
  -ftree-loop-im                         [enabled]
  -ftree-loop-ivcanon                    [enabled]
```

```
  -ftree-loop-optimize                [enabled]
  -ftree-lrs                          [disabled]
  -ftree-phiprop                      [enabled]
  -ftree-pre                          [disabled]
  -ftree-pta                          [enabled]
  -ftree-reassoc                      [enabled]
  -ftree-scev-cprop                   [enabled]
  -ftree-sink                         [disabled]
  -ftree-slp-vectorize                [enabled]
  -ftree-sra                          [disabled]
  -ftree-switch-conversion            [disabled]
  -ftree-ter                          [disabled]
  -ftree-vect-loop-version            [enabled]
  -ftree-vectorize                    [disabled]
  -ftree-vrp                          [disabled]
  -funit-at-a-time                    [enabled]
  -funroll-all-loops                  [disabled]
  -funroll-loops                      [disabled]
  -funsafe-loop-optimizations         [disabled]
  -funsafe-math-optimizations         [disabled]
  -funswitch-loops                    [disabled]
  -funwind-tables                     [disabled]
  -fvar-tracking                      [enabled]
  -fvar-tracking-assignments          [enabled]
  -fvar-tracking-assignments-toggle     [disabled]
  -fvar-tracking-uninit               [disabled]
  -fvariable-expansion-in-unroller     [disabled]
  -fvect-cost-model                   [enabled]
  -fvpt                               [disabled]
  -fweb                               [enabled]
  -fwhole-program                     [disabled]
  -fwpa                               [disabled]
  -fwrapv                             [disabled]
```

技巧

如上所示，使用 `-Q --help=optimizers` 选项可以打印出gcc的所有优化（相关的）选项，以及缺省情况下它们是否打开。类似的，你也可以查看不同优化级别下，这些优化选项是否打开：

```
$ gcc -Q --help=optimizers -O
$ gcc -Q --help=optimizers -O1
$ gcc -Q --help=optimizers -O2
$ gcc -Q --help=optimizers -O3
$ gcc -Q --help=optimizers -Og
$ gcc -Q --help=optimizers -Os
$ gcc -Q --help=optimizers -Ofast
```

详情参见gcc手册

贡献者

xmj

打印彩色诊断信息

技巧

这是gcc-4.9新增的功能，可以通过定义环境变量 `GCC_COLORS` 来彩色打印诊断信息。

也可以使用选项 `-fdiagnostics-color` 来设定。

详情参见gcc手册

贡献者

xmj

打印头文件搜索路径

例子

```
$ gcc -v foo.c
...
ignoring nonexistent directory "/usr/local/include/x86_64-linux-gnu
ignoring nonexistent directory "/usr/lib/gcc/x86_64-linux-gnu/4.6/
#include "..." search starts here:
#include <...> search starts here:
 /usr/lib/gcc/x86_64-linux-gnu/4.6/include
 /usr/local/include
 /usr/lib/gcc/x86_64-linux-gnu/4.6/include-fixed
 /usr/include/x86_64-linux-gnu
 /usr/include
End of search list.
...
```

技巧

如上所示，使用 `-v` 选项可以打印出gcc搜索头文件的路径和顺序。当然，也可以使用 `-###` 选项

贡献者

xmj

打印连接库的具体路径

例子

```
$ gcc -print-file-name=libc.a
/usr/lib/gcc/x86_64-linux-gnu/4.6/../../../x86_64-linux-gnu/libc.a
```

技巧

如上所示，使用 `-print-file-name` 选项就可以显示出gcc究竟会连接哪个libc库了。

详情参见gcc手册

贡献者

xmj

预处理

生成没有行号标记的预处理文件

技巧

有时编译程序会遇到如下类似的错误，

```
In file included from foo.c:15,
from a.h:45,
b.h:53: error: ... ...
```

如果错误是由于你所定义的一个很复杂的宏所引起的，你可能会需要先手动编译生成相应的预处理文件，查看下预处理文件中的宏扩展代码。比如，先运行

```
gcc -E foo.c -o foo.i
```

来生成foo.i预处理文件。然后，还可以尝试手动修改、编译这个预处理文件。

但是，由于生成的预处理文件中含有行号标记（linemarker），所以，运行

```
gcc -c foo.i -o foo.o
```

所得到的错误行号信息还是跟最初的一样，如果可以将预处理文件中的行号标记都去掉，似乎会有些帮助。

幸好，gcc提供了这个选项：

> -P Inhibit generation of linemarkers in the output from the preprocessor. This might be useful when running the preprocessor on something that is not C code, and will be sent to a program which might be confused by the linemarkers.

运行

```
gcc -E -P foo.c -o foo.i
```

即可。

详情参见gcc手册

贡献者

xmj

在命令行中预定义宏

例子

```
#include <stdio.h>

int main (void)
{
  int i, sum;

  for (i = 1, sum = 0; i <= 10; i++)
    {
      sum += i;
    #ifdef DEBUG
      printf ("sum += %d is %d\n", i, sum);
    #endif
    }
  printf ("total sum is %d\n", sum);

  return 0;
}
```

技巧

使用 `-D` 选项可以在命令行中预定义一个宏，比如：

```
$ gcc -D DEBUG macro.c
```

中间可以没有空格：

```
$ gcc -DDEBUG macro.c
```

详情参见gcc手册

贡献者

xmj

在命令行中取消宏定义

技巧

类似于 `-D` 选项，你可以使用 `-U` 选项在命令行中取消一个宏的定义，比如：

```
$ gcc -U DEBUG macro.c
```

中间可以没有空格：

```
$ gcc -UDEBUG macro.c
```

详情参见gcc手册

贡献者

xmj

汇编

把选项传给汇编器

例子

```
#include <stdio.h>

int main(void)
{
  int i;

  for (i = 0; i < 10; i++)
    printf("%d ", i);
  putchar ('\n');

  return 0;
}
```

技巧

使用`-Wa,option`可以将选项`option`传递给汇编器。

注意，逗号和选项之间不能有空格。例如：

```
$ gcc -c -Wa,-L foo.c
$ objdump -d foo.o

foo.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <main>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   48 83 ec 10             sub    $0x10,%rsp
   8:   c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%rbp)
   f:   eb 1b                   jmp    2c <.L2>

0000000000000011 <.L3>:
  11:   b8 00 00 00 00          mov    $0x0,%eax
  16:   8b 55 fc                mov    -0x4(%rbp),%edx
  19:   89 d6                   mov    %edx,%esi
  1b:   48 89 c7                mov    %rax,%rdi
  1e:   b8 00 00 00 00          mov    $0x0,%eax
  23:   e8 00 00 00 00          callq  28 <.L3+0x17>
  28:   83 45 fc 01             addl   $0x1,-0x4(%rbp)

000000000000002c <.L2>:
  2c:   83 7d fc 09             cmpl   $0x9,-0x4(%rbp)
  30:   7e df                   jle    11 <.L3>
  32:   bf 0a 00 00 00          mov    $0xa,%edi
  37:   e8 00 00 00 00          callq  3c <.L2+0x10>
  3c:   b8 00 00 00 00          mov    $0x0,%eax
  41:   c9                      leaveq
  42:   c3                      retq
```

这里的 `-L` 是汇编器as的选项，用于在目标文件中保留局部符号（local symbol）。可以看到，反汇编代码中给出了每个局部符号。

如果此时你使用 `oprofile` 来统计性能事件，那么获得的结果将不是以函数为单位了，而是以这些符号所划分的代码块为单位。

详情参见gcc手册和as手册

贡献者

xmj

生成有详细信息的汇编文件

例子

```
#include <stdio.h>

int main(void)
{
  int i;

  for (i = 0; i < 10; i++)
    printf("%d ", i);
  putchar ('\n');

  return 0;
}
```

技巧

使用 `-fverbose-asm` 选项就可以生成带有详细信息的汇编文件：

```
$ gcc -S -fverbose-asm foo.c
$ cat foo.s
    .file    "foo.c"
# GNU C (Ubuntu/Linaro 4.6.3-1ubuntu5) version 4.6.3 (x86_64-linux-
#    compiled by GNU C version 4.6.3, GMP version 5.0.2, MPFR versi
# GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsi
# options passed:  -imultilib . -imultiarch x86_64-linux-gnu foo.c
# -mtune=generic -march=x86-64 -fverbose-asm -fstack-protector
# options enabled:  -fasynchronous-unwind-tables -fauto-inc-dec
# -fbranch-count-reg -fcommon -fdelete-null-pointer-checks -fdwarf2
# -fearly-inlining -feliminate-unused-debug-types -ffunction-cse -f
# -fident -finline-functions-called-once -fira-share-save-slots
# -fira-share-spill-slots -fivopts -fkeep-static-consts
# -fleading-underscore -fmath-errno -fmerge-debug-strings
# -fmove-loop-invariants -fpeephole -fprefetch-loop-arrays
# -freg-struct-return -fsched-critical-path-heuristic
# -fsched-dep-count-heuristic -fsched-group-heuristic -fsched-inter
# -fsched-last-insn-heuristic -fsched-rank-heuristic -fsched-spec
# -fsched-spec-insn-heuristic -fsched-stalled-insns-dep -fshow-colu
# -fsigned-zeros -fsplit-ivs-in-unroller -fstack-protector
# -fstrict-volatile-bitfields -ftrapping-math -ftree-cselim -ftree-
# -ftree-loop-if-convert -ftree-loop-im -ftree-loop-ivcanon
# -ftree-loop-optimize -ftree-parallelize-loops= -ftree-phiprop -ft
# -ftree-reassoc -ftree-scev-cprop -ftree-slp-vectorize
```

```
# -ftree-vect-loop-version -funit-at-a-time -funwind-tables
# -fvect-cost-model -fverbose-asm -fzero-initialized-in-bss
# -m128bit-long-double -m64 -m80387 -maccumulate-outgoing-args
# -malign-stringops -mfancy-math-387 -mfp-ret-in-387 -mglibc -mieee
# -mmmx -mno-sse4 -mpush-args -mred-zone -msse -msse2 -mtls-direct-

# Compiler executable checksum: 75e879ed14f91af504f4150eadeaa0e6

    .section    .rodata
.LC0:
    .string    "%d "
    .text
    .globl    main
    .type    main, @function
main:
.LFB0:
    .cfi_startproc
    pushq    %rbp    #
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp    #,
    .cfi_def_cfa_register 6
    subq    $16, %rsp    #,
    movl    $0, -4(%rbp)    #, i
    jmp    .L2    #
.L3:
    movl    $.LC0, %eax    #, D.2049
    movl    -4(%rbp), %edx    # i, tmp62
    movl    %edx, %esi    # tmp62,
    movq    %rax, %rdi    # D.2049,
    movl    $0, %eax    #,
    call    printf    #
    addl    $1, -4(%rbp)    #, i
.L2:
    cmpl    $9, -4(%rbp)    #, i
    jle    .L3    #,
    movl    $10, %edi    #,
    call    putchar    #
    movl    $0, %eax    #, D.2050
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size    main, .-main
    .ident    "GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3"
    .section    .note.GNU-stack,"",@progbits
```

可以看到，在汇编文件中给出了gcc所使用的具体选项，以及汇编指令操作数所对应的源程序（或中间代码）中的变量。

详情参见gcc手册

贡献者

xmj

调试

利用Address Sanitizer工具检查内存访问错误

例子

```
a.c:
#include <stdio.h>

int main(void) {
        // your code goes here
        int a[3] = {0};
        a[3] = 1;

        printf("%d\n", a[3]);
        return 0;
}

b.c:
#include <stdio.h>
#include <malloc.h>

int main(void) {
        int *p = NULL;

        p = malloc(10 * sizeof(int));
        free(p);
        *p = 3;
        return 0;
}
```

技巧

gcc从`4.8`版本起，集成了`Address Sanitizer`工具，可以用来检查内存访问的错误（编译时指定“`-fsanitize=address`”）。以上面`a.c`程序为例：

```
gcc -fsanitize=address -g -o a a.c
```

执行`a`程序：

```
[root@localhost nan]# ./a
=================================================================
==539==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x
WRITE of size 4 at 0x7fff3a152c9c thread T0
    #0 0x4009b5 in main /home/nan/a.c:6
    #1 0x34e421ed1c in __libc_start_main (/lib64/libc.so.6+0x34e421
    #2 0x4007b8 (/home/nan/a+0x4007b8)

Address 0x7fff3a152c9c is located in stack of thread T0 at offset 4
    #0 0x400907 in main /home/nan/a.c:3

  This frame has 1 object(s):
    [32, 44) 'a' <== Memory access at offset 44 overflows this vari
HINT: this may be a false positive if your program uses some custom
      (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow /home/nan/a.c:6 ma
Shadow bytes around the buggy address:
  0x100067422540: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100067422550: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100067422560: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100067422570: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100067422580: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1
=>0x100067422590: f1 f1 00[04]f4 f4 f3 f3 f3 f3 00 00 00 00 00 00
  0x1000674225a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000674225b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000674225c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000674225d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000674225e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Shadow byte legend (one shadow byte represents 8 application bytes)
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Heap right redzone:      fb
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack partial redzone:   f4
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Contiguous container OOB:fc
  ASan internal:           fe
==539==ABORTING
```

可以看到，执行程序时检测出了 `a` 数组的越界访问（ `a[3] = 1` ）。

再看一下 `b` 程序：

```
gcc -fsanitize=address -g -o b b.c
```

执行`b`程序：

```
[root@localhost nan]# ./b
=================================================================
==1951==ERROR: AddressSanitizer: heap-use-after-free on address 0x6
WRITE of size 4 at 0x60400000dfd0 thread T0
    #0 0x4007f8 in main /home/nan/b.c:9
    #1 0x34e421ed1c in __libc_start_main (/lib64/libc.so.6+0x34e42
    #2 0x400658 (/home/nan/b+0x400658)

0x60400000dfd0 is located 0 bytes inside of 40-byte region [0x6040
freed by thread T0 here:
    #0 0x7fbbb7a7d057 in __interceptor_free /opt/gcc-4.9.2/src/gcc-
    #1 0x4007c1 in main /home/nan/b.c:8
    #2 0x34e421ed1c in __libc_start_main (/lib64/libc.so.6+0x34e42

previously allocated by thread T0 here:
    #0 0x7fbbb7a7d26f in __interceptor_malloc /opt/gcc-4.9.2/src/g
    #1 0x4007b1 in main /home/nan/b.c:7
    #2 0x34e421ed1c in __libc_start_main (/lib64/libc.so.6+0x34e42

SUMMARY: AddressSanitizer: heap-use-after-free /home/nan/b.c:9 mai
Shadow bytes around the buggy address:
  0x0c087fff9ba0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c087fff9bb0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c087fff9bc0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c087fff9bd0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c087fff9be0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x0c087fff9bf0: fa fa fa fa fa fa fa fa fa fa[fd]fd fd fd fd fa
  0x0c087fff9c00: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c087fff9c10: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c087fff9c20: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c087fff9c30: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c087fff9c40: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Heap right redzone:      fb
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack partial redzone:   f4
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Contiguous container OOB:fc
  ASan internal:           fe
==1951==ABORTING
```

执行程序时检测出了访问释放内存的错误（`*p = 3`）。
详情参见gcc手册

贡献者

nanxiao

利用Thread Sanitizer工具检查数据竞争的问题

例子

```
#include <pthread.h>
int Global;
void *Thread1(void *x) {
  Global = 42;
  return x;
}
int main(void) {
  pthread_t t;
  pthread_create(&t, NULL, Thread1, NULL);
  Global = 43;
  pthread_join(t, NULL);
  return Global;
}
```

技巧

gcc从`4.8`版本起，集成了`Address Sanitizer`工具，可以用来检查数据竞争的问题（编译时指定“`-fsanitize=thread -fPIE -pie`”）。以上面程序为例：

```
gcc -fsanitize=thread -fPIE -pie -g -o a a.c -lpthread
```

执行`a`程序：

```
[root@localhost nan]# ./a
==================
WARNING: ThreadSanitizer: data race (pid=14545)
  Write of size 4 at 0x7f055b4802b0 by thread T1:
    #0 Thread1 /home/nan/a.c:4 (a+0x000000000a87)

  Previous write of size 4 at 0x7f055b4802b0 by main thread:
    #0 main /home/nan/a.c:10 (a+0x000000000ae8)

  Location is global 'Global' of size 4 at 0x7f055b4802b0 (a+0x0000

  Thread T1 (tid=14547, running) created by main thread at:
    #0 pthread_create /opt/gcc-4.9.2/src/gcc-4.9.2/libsanitizer/tsa
    #1 main /home/nan/a.c:9 (a+0x000000000ad9)

SUMMARY: ThreadSanitizer: data race /home/nan/a.c:4 Thread1
==================
ThreadSanitizer: reported 1 warnings
```

可以看到，执行程序时检测出了对`Global`变量的竞争访问。
详情参见gcc手册

贡献者

nanxiao

连接

把选项传给连接器

例子

```
#include <stdio.h>

int main (void)
{
  puts ("Hello world!");
  return 0;
}
```

技巧

使用 `-Wl,option` 可以将选项 `option` 传递给连接器。

注意，逗号和选项之间不能有空格。一种常见用法，就是让连接器生成内存映射文件，例如：

```
$ gcc -Wl,-Map=output.map foo.c
$ cat output.map
Archive member included because of file (symbol)

/usr/lib/x86_64-linux-gnu/libc_nonshared.a(elf-init.oS)
                              /usr/lib/gcc/x86_64-linux-gnu/4.6/../

Discarded input sections

 .note.GNU-stack
                0x0000000000000000        0x0 /usr/lib/gcc/x86_64-l
 .gnu_debuglink
                0x0000000000000000        0xc /usr/lib/gcc/x86_64-l
 .note.GNU-stack
                0x0000000000000000        0x0 /usr/lib/gcc/x86_64-l
 .gnu_debuglink
                0x0000000000000000        0xc /usr/lib/gcc/x86_64-l
 .note.GNU-stack
                0x0000000000000000        0x0 /usr/lib/gcc/x86_64-l
 .note.GNU-stack
                0x0000000000000000        0x0 /tmp/ccBOhdmq.o
 .note.GNU-stack
                0x0000000000000000        0x0 /usr/lib/x86_64-linux
 .note.GNU-stack
                0x0000000000000000        0x0 /usr/lib/gcc/x86_64-l
 .note.GNU-stack
```

```
                 0x0000000000000000        0x0 /usr/lib/gcc/x86_64-l
 .gnu_debuglink
                 0x0000000000000000        0xc /usr/lib/gcc/x86_64-l

Memory map

 ** file header
                 0x0000000000400000       0x40
 ** segment headers
                 0x0000000000400040      0x1f8

.interp          0x0000000000400238       0x1c
 ** fill         0x0000000000400238       0x1c

.note.ABI-tag    0x0000000000400254       0x20
 .note.ABI-tag   0x0000000000400254       0x20 /usr/lib/gcc/x86_64-l

.note.gnu.build-id
                 0x0000000000400274       0x24
 ** note header
                 0x0000000000400274       0x10
 ** zero fill    0x0000000000400284       0x14

.dynsym          0x0000000000400298       0x78
 ** dynsym       0x0000000000400298       0x78

.dynstr          0x0000000000400310       0x51
 ** string table
                 0x0000000000400310       0x51

.gnu.hash        0x0000000000400368       0x1c
 ** hash         0x0000000000400368       0x1c

.gnu.version     0x0000000000400384        0xa
 ** versions     0x0000000000400384        0xa

.gnu.version_r   0x0000000000400390       0x20
 ** version refs
                 0x0000000000400390       0x20

.rela.dyn        0x00000000004003b0       0x18
 ** dynamic relocs
                 0x00000000004003b0       0x18

.rela.plt        0x00000000004003c8       0x30
 ** dynamic relocs
                 0x00000000004003c8       0x30

.init            0x00000000004003f8       0x18
 .init           0x00000000004003f8        0x9 /usr/lib/gcc/x86_64-l
                 0x00000000004003f8                _init
 .init           0x0000000000400401        0x5 /usr/lib/gcc/x86_64-l
 .init           0x0000000000400406        0x5 /usr/lib/gcc/x86_64-l
```

```
 .init          0x000000000040040b       0x5 /usr/lib/gcc/x86_64-l

.plt            0x0000000000400410      0x30
 ** PLT         0x0000000000400410      0x30

.text           0x0000000000400440     0x1d8
 .text          0x0000000000400440      0x2c /usr/lib/gcc/x86_64-l
                0x0000000000400440                _start
 .text          0x000000000040046c      0x17 /usr/lib/gcc/x86_64-l
 ** fill        0x0000000000400483       0xd
 .text          0x0000000000400490      0x92 /usr/lib/gcc/x86_64-l
 .text          0x0000000000400522      0x15 /tmp/ccBOhdmq.o
                0x0000000000400522                main
 ** fill        0x0000000000400537       0x9
 .text          0x0000000000400540      0x92 /usr/lib/x86_64-linux
                0x0000000000400540                __libc_csu_init
                0x00000000004005d0                __libc_csu_fini
 ** fill        0x00000000004005d2       0xe
 .text          0x00000000004005e0      0x36 /usr/lib/gcc/x86_64-l
 ** fill        0x0000000000400616       0x2
 .text          0x0000000000400618       0x0 /usr/lib/gcc/x86_64-l

.fini           0x0000000000400618       0xe
 .fini          0x0000000000400618       0x4 /usr/lib/gcc/x86_64-l
                0x0000000000400618                _fini
 .fini          0x000000000040061c       0x5 /usr/lib/gcc/x86_64-l
 .fini          0x0000000000400621       0x5 /usr/lib/gcc/x86_64-l

.rodata         0x0000000000400628      0x11
 ** merge constants
                0x0000000000400628       0x4
 .rodata        0x000000000040062c       0xd /tmp/ccBOhdmq.o

.eh_frame       0x0000000000400640      0xa4
 ** eh_frame    0x0000000000400640      0xa0
 .eh_frame      0x00000000004006e0       0x4 /usr/lib/gcc/x86_64-l

.eh_frame_hdr   0x00000000004006e4      0x2c
 ** eh_frame_hdr
                0x00000000004006e4      0x2c

.ctors          0x0000000000401e28      0x10
 .ctors         0x0000000000401e28       0x8 /usr/lib/gcc/x86_64-l
 .ctors         0x0000000000401e30       0x8 /usr/lib/gcc/x86_64-l

.dtors          0x0000000000401e38      0x10
 .dtors         0x0000000000401e38       0x8 /usr/lib/gcc/x86_64-l
 .dtors         0x0000000000401e40       0x8 /usr/lib/gcc/x86_64-l
                0x0000000000401e40                __DTOR_END__

.jcr            0x0000000000401e48       0x8
 .jcr           0x0000000000401e48       0x0 /usr/lib/gcc/x86_64-l
 .jcr           0x0000000000401e48       0x8 /usr/lib/gcc/x86_64-l
```

```
.dynamic         0x0000000000401e50      0x190
 ** dynamic      0x0000000000401e50      0x190

.got             0x0000000000401fe0        0x8
 ** GOT          0x0000000000401fe0        0x8

.got.plt         0x0000000000401fe8       0x28
 ** GOT PLT      0x0000000000401fe8       0x28
 ** GOT IRELATIVE PLT
                 0x0000000000402010        0x0
 ** GOT          0x0000000000402010        0x0

.data            0x0000000000402010       0x10
 .data           0x0000000000402010        0x4 /usr/lib/gcc/x86_64-l
                 0x0000000000402010                data_start
                 0x0000000000402010                __data_start
 .data           0x0000000000402014        0x0 /usr/lib/gcc/x86_64-l
 .data           0x0000000000402018        0x8 /usr/lib/gcc/x86_64-l
                 0x0000000000402018                __dso_handle
 .data           0x0000000000402020        0x0 /tmp/ccBOhdmq.o
 .data           0x0000000000402020        0x0 /usr/lib/x86_64-linux
 .data           0x0000000000402020        0x0 /usr/lib/gcc/x86_64-l
 .data           0x0000000000402020        0x0 /usr/lib/gcc/x86_64-l

.bss             0x0000000000402020       0x10
 .bss            0x0000000000402020        0x0 /usr/lib/gcc/x86_64-l
 .bss            0x0000000000402020        0x0 /usr/lib/gcc/x86_64-l
 .bss            0x0000000000402020       0x10 /usr/lib/gcc/x86_64-l
 .bss            0x0000000000402030        0x0 /tmp/ccBOhdmq.o
 .bss            0x0000000000402030        0x0 /usr/lib/x86_64-linux
 .bss            0x0000000000402030        0x0 /usr/lib/gcc/x86_64-l
 .bss            0x0000000000402030        0x0 /usr/lib/gcc/x86_64-l

.comment         0x0000000000000000       0x2b
 ** merge strings
                 0x0000000000000000       0x2b

.note.gnu.gold-version
                 0x0000000000000000       0x1c
 ** note header
                 0x0000000000000000       0x10
 ** fill         0x0000000000000010        0x9
 ** zero fill    0x0000000000000019        0x3

.symtab          0x0000000000000000      0x390
 ** symtab       0x0000000000000000      0x390

.strtab          0x0000000000000000      0x1d5
 ** string table
                 0x0000000000000000      0x1d5

.shstrtab        0x0000000000000000      0x115
```

```
  ** string table
                    0x0000000000000000      0x115
```

详情参见gcc手册

贡献者

xmj

设置动态连接器

技巧

有人问我，如何通过选项来指定动态连接器，而不使用缺省系统自带的动态连接器。我后来查了下ld的手册，有这么一个选项：

```
-Ifile
--dynamic-linker=file
    Set the name of the dynamic linker. This is only meaningful whe
```

看起来，可以通过如下方式来完成：

```
$ gcc foo.c -Wl,-I/home/xmj/tmp/ld-2.15.so
$ ldd a.out
linux-vdso.so.1 =>  (0x00007fffce5fe000)
/usr/local/lib/libtrash.so (0x00007f1980477000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f19800a3000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f197fe9e000)
/home/xmj/tmp/ld-2.15.so => /lib64/ld-linux-x86-64.so.2 (0x00007f19
```

注意，tmp目录下的动态连接器因为也是动态连接的，所以它本身是依赖系统缺省的动态连接器。

详情参见ld手册

贡献者

xmj

函数属性

禁止函数被优化掉

例子

```
#if (GCC_VERSION > 4000)
#define DEBUG_FUNCTION __attribute__ ((__used__))
#define DEBUG_VARIABLE __attribute__ ((__used__))
#else
#define DEBUG_FUNCTION
#define DEBUG_VARIABLE
#endif

DEBUG_FUNCTION void
debug_bb (basic_block bb)
{
  dump_bb (bb, stderr, 0);
}
```

技巧

上面的例子是gcc的源码。使用gcc的扩展功能——函数属性 `__attribute__ ((__used__))` ，可以指定该函数是有用的，不能被优化掉。

详情参见gcc手册

贡献者

xmj

强制函数永远以inline的形式调用

例子

```
#if defined(__GNUC__)
#define FORCEDINLINE  __attribute__((always_inline))
#else
#define FORCEDINLINE
#endif

FORCEDINLINE int add(int a,int b)
{
  return a+b;
}
```

技巧

上面的例子是gcc的源码。使用gcc的扩展功能——函数属性`__attribute__ ((always_inline))`，可以指定该函数永远以inline的形式调用

详情参见gcc手册

贡献者

mengke

常见错误

error: cast from ... to ... loses precision

例子

```
#include <iostream>

class Foo {
 public:
  void print() const {
    std::cout << (int)(this) << "\n";
  }
};

int main()
{
  class Foo foo;

  foo.print();
  return 0;
}
```

技巧

在g++编译上面的例子，会报如下错误：

```
$ g++ foo.cc
foo.cc: In member function ‘void Foo::print() const’:
foo.cc:6:28: error: cast from ‘const Foo*’ to ‘int’ loses precision
```

这是一个强制类型转换的错误，你可以修改源代码为：

```
std::cout << (int*)(this) << "\n";
```

即可。

如果，你不想（或不能）去修改源程序，只是应为升级了gcc而带来了这样的错误，那么也可以使用 `-fpermissive` 选项，将错误降低为警告：

```
$ g++ foo.cc -fpermissive
foo.cc: In member function ‘void Foo::print() const’:
foo.cc:6:28: warning: cast from ‘const Foo*’ to ‘int’ loses precisi
```

详情参见gcc手册

贡献者

xmj

all warnings being treated as errors

技巧

在ubuntu系统下编译一个程序包，有时会遇到这样的错误：

```
$ make
...
cc1: all warnings being treated as errors
```

这是因为缺省的CFLAGS里含有 `-Werror` 选项，将警告信息升级为错误。当然，一方面这可以让你重视这些可能会带来隐患的警告信息；但，如果你不想修改源码，也可以把这个选项关掉，通过修改Makefile或者使用命令行：

```
$ make CFLAGS="... -Wno-error"
```

详情参见gcc手册

贡献者

xmj

其它

只做语法检查

例子

```
$ cat foo.c
union u {
  char c;
  int i;
}
$ gcc -fsyntax-only foo.c
foo.c:4:1: error: expected identifier or '(' at end of input
```

技巧

如上所示，使用`-fsyntax-only`选项可以只做语法检查，不进行实际的编译输出。

详情参见gcc手册

贡献者

xmj

保存临时文件

例子

```
$ gcc -save-temps a/foo.c
$ ls foo.*
foo.c  foo.i  foo.o  foo.s

$ gcc -save-temps=obj a/foo.c -o a/foo
$ ls a
foo  foo.c  foo.i  foo.o  foo.s
```

技巧

如上所示，使用选项 `-save-temps` 可以保存gcc运行过程中生成的临时文件。这些中间文件的名字是基于源文件而来，并且保存在当前目录下。

如果你在不同目录下有重名的源文件，那么中间文件就会有冲突了。此时，你可以使用 `-save-temps=obj` 来指定中间文件名基于目标文件而定，并保存在目标文件所在目录下。

详情参见gcc手册

贡献者

xmj

打开警告信息

技巧

你的程序编译通过了，但并不意味着已经万事大吉，也许还存在一些不规范的地方，或者一些错误隐患。建议，使用 `-Wall` 选项打开所有的警告信息，把所有的警告都处理掉。

```
$ gcc -Wall ...
```

详情参见gcc手册

贡献者

xmj

指定语言类型

技巧

gcc是通过文件名后缀来判断源代码语言类型的。

如果你从标准输入把源码传给gcc，那么就需要通过 `-x` 选项显式的指定语言类型：

```
$ echo "int x;" | gcc -S -x c -
$ cat ./-.s
    .file    ""
    .comm    x,4,4
    .ident    "GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3"
    .section    .note.GNU-stack,"",@progbits
```

详情参见gcc手册

贡献者

xmj

改变结构体成员的字节对齐

例子

```
#include <stdio.h>

typedef struct
{
        char a;
        int b;
} ST_A;

int main(void)
{
        printf("sizeof(ST_A)=%ld\n",sizeof(ST_A));
}
```

技巧

在上面的程序里，`ST_A`结构体的内存布局默认是这样的：

Offset	1byte	1byte	1byte	1byte
0	a	填充字节	填充字节	填充字节
4	b	b	b	b

编译执行，结果如下：

```
root@ubuntu:~$ gcc -g -o a a.c
root@ubuntu:~$ ./a
sizeof(ST_A)=8
```

使用gcc的"`-fpack-struct[=n]`"选项（“`n`”需要为`2`的倍数）可以改变成员的地址对齐。例如指定“`n=2`”时，将标明结构体成员的最大对齐地址为2。这样`ST_A`结构体中的成员`b`的地址将不再按照`4`字节对齐，内存布局变为：

Offset	1byte	1byte	1byte	1byte
0	a	填充字节	b	b
4	b	b		

编译执行，结果如下：

```
root@ubuntu:~$ gcc -g -fpack-struct=2 -o a a.c
root@ubuntu:~$ ./a
sizeof(ST_A)=6
```

当不指定“`n`”时，将没有填充字节，所有成员将一个挨着一个排在一起：

Offset	1byte	1byte	1byte	1byte
0	a	b	b	b
4	b			

编译执行，结果如下：

```
root@ubuntu:~$ gcc -g -fpack-struct -o a a.c
root@ubuntu:~$ ./a
sizeof(ST_A)=5
```

由于这个编译选项会导致ABI(Application Binary Interface)的改变，所以使用时一定要谨慎。 详情参见gcc手册

贡献者

nanxiao